VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Analýza jízdních pruhů pomocí obrazu

Lane detection using computer vision

2018

Jan Tyc

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**Jan Tyc**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	Analýza jízdních pruhů za pomocí obrazů Lane Detection Using Computer Vision
Jazyk vypracování:	čeština

Zásady pro vypracování:

Autonomní řízení vozidel se v posledních letech stává zajímavým a stále více citovaným tématem. Vozidla vybavená takovým systémem musí umět za pomocí různých senzorů detekovat a rozpoznat celou řadu zájmových objektů na silnicích. Například protijedoucí vozidla, chodce, dopravní značky nebo jízdní pruhy.

1. Seznamte se se základními přístupy z oblasti analýzy tvarů objektů v obrazech s cílem využít tyto metody k analýze jízdních pruhů.
2. Seznamte se s vhodnou knihovnou pro s práci s obrazy (například OpenCV, Dlib).
3. S pomocí nastudovaných přístupů jedno vhodné řešení pro analýzu jízdních pruhů implementujte.
4. Experimentálně ověřte funkčnost, přesnost a rychlost navrženého řešení.
5. Své závěry řádně zdokumentujte v textu práce.

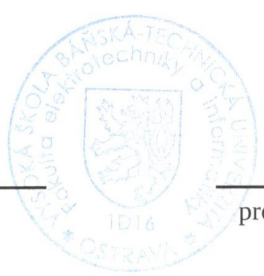
Seznam doporučené odborné literatury:

[1] J. Kim, M. Lee. Robust Lane Detection Based on Convolutional Neural Network and Random Sample Consensus, Lecture Notes in Computer Science, vol. 8834, Springer, pp. 454-461, 2014
[2] J. Li, X. Mei, D. Prokhorov and D. Tao. Deep Neural Network for Structural Prediction and Lane Detection in Traffic Scene, in IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 3, pp. 690-703, 2017

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radovan Fusek, Ph.D.**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 ..... Jan Tax .....

Chtěl bych poděkovat vedoucímu mé práce Radovanu Fuskovi za ochotnou pomoc a dobré rady.

Abstrakt

Autonomní řízení vozidel se v posledních letech stává zajímavým a stále více citovaným tématem. Vozidla vybavená takovým systémem musí umět za pomocí různých senzorů detekovat a rozpoznat celou řadu zájmových objektů na silnicích. Například protijedoucí vozidla, chodce, dopravní značky nebo jízdní pruhy.

Cílem práce je vytvoření programu, který dokáže rozpoznat jízdní pruh a do obrázku nebo videa vykreslit oblast, která odpovídá tomuto jízdnímu pruhu.

Klíčová slova: Zpracování obrazu, strojové vidění, detekce jízdních pruhů, OpenCV

Abstract

Autonomous car driving is recently becoming more interesting and more cited topic. Vehicles equiped with such system have to be able to detect and recognize various objects on the road by different sensors. For example oncomming traffic, pedestrians, traffic signs or traffic lanes.

Goal of this work is to create a program, which is able to recognize traffic lane and draw an area into an image or a video accordingly to the traffic lane as accurately as possible.

Key Words: Image processing, computer vision, traffic lane detection, OpenCV

Obsah

Seznam použitých zkratek a symbolů

MLP	–	Multi-layer Perceptron
HOG	–	Histogram of Oriented Gradients
SIFT	–	Scale-invariant Feature Transform
YOLO	–	You Only Look Once
RCNN	–	Regional Convolutional Neural Network
SVM	–	Support Vetor Machine
SURF	–	Speeded Up Robust Features

Seznam obrázků

Seznam tabulek

1 Úvod

V dnešní době je stále populárnější téma autonomní jízdy. Mnohé automobilky se snaží o vytvoření takového vozidla, které je schopno samo a bezpečně jezdit bez jakéhokoliv lidského zásahu. Někteří výrobci už takové technologie ve svých autech používají. Například Mercedes či BMW tyto technologie používá pro detekci překážek, nebo chytrá světla. Automobilka Tesla dokonce už vyvinula svého autopilota, který dokáže sám jezdit, avšak zatím jen po udržovaných cestách, kde jsou dobře vidět jízdní pruhy.

Tato problematika v sobě zahrnuje spoustu technologií včetně zpracování obrazu. Tyto vozidla musí umět detekovat jízdní pruhy, značky, chodce, další vozidla v provozu a mnoho dalších věcí. Existuje už mnoho způsobů jak detekovat objekty v obraze. Klasickými algoritmy (detekce hran, HOG, SIFT...) strojovým učením počínaje a hlubokým učením konče.

V této práci budu nejprve popisovat existující způsoby detekce jízdních pruhů. Poté jak si upravit vstupní data tak, aby se odstranil šum a zlepšila se tak přesnost při práci s těmito daty. V další kapitole bude vysvětleno co je to hrana a jak se v obraze detekují hrany, což je základní operací při zpracovávání obrazu. Další kapitola bude o Houghově transformaci, která slouží k detekci přímek v obraze. Dále si popíšeme, jaké knihovny dnes existují pro práci s obrazem. Další kapitola bude pojednávat o vykreslování oblasti jízdního pruhu do obrazu. Následovat bude kapitola o mém implementovaném řešení, ve které budu popisovat jaké metody a postupy jsem si vybral k vyřešení mého zadání. Dále bude testování mnou navrženého řešení na konkrétních příkladech obrázků a na závěr stručně shrnu své řešení.

2 Existující přístupy

Detekce jízdních pruhů je v poslední době hodně řešeným tématem díky novému trendu autonomních vozidel. Je tedy více než zřejmé, že se skoro každá lepší automobilka snaží do svých vozidel zahrnout alespoň nějaké asistenční systémy, mezi které například patří systém udržování jízdy v daném pruhu.

Existuje tedy více postupů k detekci jízdního pruhu na vozovce. V této práci se prakticky zaměřím na detekci bez učení.

2.1 Detekce bez učení

Jedním možným způsobem detekce jízdních pruhů, je detekce bez učení. Existuje mnoho různých způsobů jak detekovat jízdní pruh bez strojového učení. Některé metody jsou založené na detekci hran a přímek, některé zase na transformaci obrazu. Jiné metody zase pracují s informacemi o světle nebo barvě.

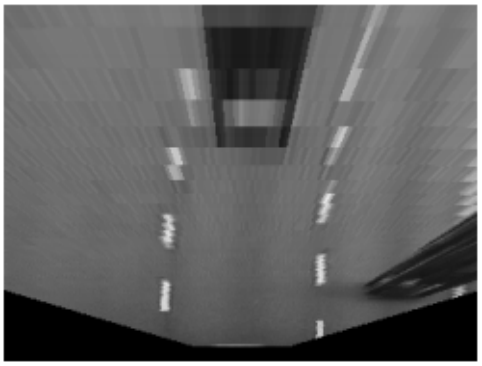
Jedním z těch robustnějších přístupů je detekce za pomocí inverzního perspektivního mapování. Při snímání cesty kamerou je výsledkem obraz okolí. Tento obraz je však ovlivněn mnoha faktory, například úhlem, pod kterým byl obraz pořízen. Z obrazu cesty pořízeného v autě lze jednoduše vidět, že má perspektivní efekt (čáry na cestě se sbíhají do jednoho bodu). Metoda inverzního perspektivního mapování je založena na převodu ze středového do ortogonálního promítání (ptačí perspektivy), což znamená, že vzdálenost všech bodů v obraze od kamery je stejná. V ideálním případě by měly být známy informace jako například úhel kamery či vzdálenost od cíle, což jsou klíčové informace k tomu, aby bylo možné promítnout obraz do ptačí perspektivy. Při natáčení v autě však tyto informace jednoduše zjistit nelze, ale dají se z obrazu dopočítat. Po transformaci do ptačí perspektivy se pak hledá samotný jízdní pruh a to pomocí vertikálních a horizontálních histogramů. Výhody tohoto přístupu jsou zejména přesnost a robustnost. Nevýhodou je však vyšší výpočetní náročnost. Příklad použití této metody je ukázán na obrázku 1. O tomto způsobu pojednává publikace [3] a [2].

Další způsob detekce jízdních pruhů je založen na detekci hran a následného použití Houghovy transformace pro detekci přímek. První krok této metody je detekce hran z vyfiltrovaného obrazu. Poté se provede Houghova transformace, která aproximuje křivky podle hranových bodů. Jednodušší metody detekují tyto křivky pomocí přímek, které dobře fungují na rovných cestách. Komplexnější metody pak dokáží správě detekovat i zatáčky, kde silniční čáry už nejsou jednoduchou přímkou. Takový způsob je popsán v publikaci [8] a znázorněn na obrázku 2.

Má práce bude založena na této metodě. V mé práci nebudu pouze vykreslovat detekované jízdní pruhy, ale přidám také vykreslení segmentu, který odpovídá tomuto jízdnímu pruhu. Cílem mého řešení je i funkčnost na mírných zatáčkách.

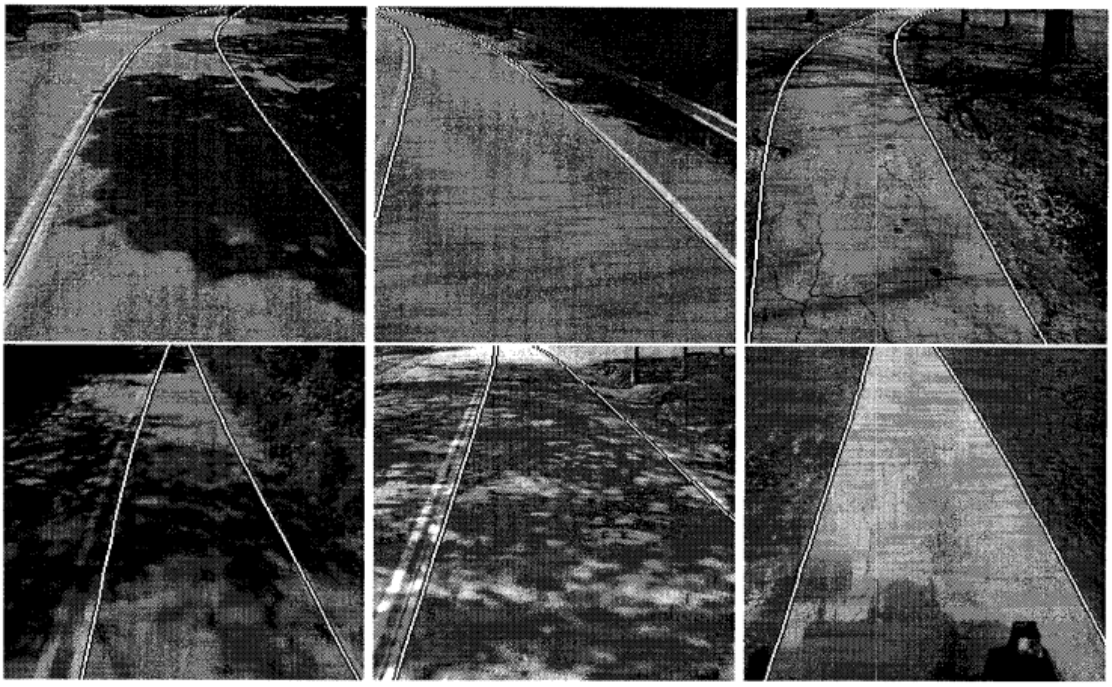
Existují i způsoby detekce jízdních pruhů založené na práci s informacemi o barvě a světle. Tyto metody fungují na základě segmentace obrazu podle barvy, jelikož barva silnic je v drtivé většině případů přibližně stejná. U tohoto způsobu se nejdříve provádí shlukování pixelů, které

(a) Původní obraz před inverzním perspektivním mapováním [2]



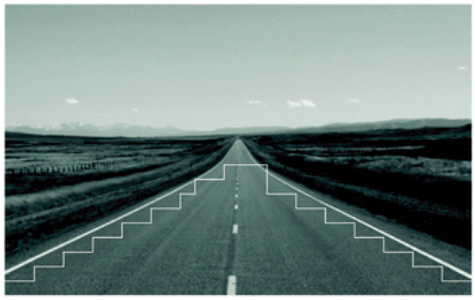
(b) Výsledný transformovaný obraz [2]

Obrázek 1: Ukázka metody inverzního perspektivního mapování



Obrázek 2: Výsledky detekce pomocí Houghovy transformace.[8]

Obrázek 3: Výsledky detekce pomocí segmentace barev [10]



(a) Výběr oblasti silnice[11]

(b) Ohraničená silnice[11]

Obrázek 4: Ukázka detekce silnice pomocí textury

jsou navzájem podobné. Tímto vzniknou oblasti, které mají podobnou barvu a ty se následně označí jako silnice nebo jízdní pruh. Pro tyto účely se hodí bilaterální filtr. Tato metoda obecně funguje dobře i na nestrukturovaných a nefrekventovaných cestách, které neobsahují žádné silniční značení. Publikace, která se zabývá touto problematikou je [9]. Existuje však i česká práce na toto téma [10]. Příklad takového přístupu je na obrázku 3.

Podobná metoda segmentace obrazu podle barvy je segmentace podle textury. Tento přístup je založen na práci s histogramy. Základem je myšlenka, že se obraz skládá ze tří částí. Ze silnice, blízkého okolí a dalekého okolí. Nejprve se naleznou oblasti, které nejlépe odpovídají těmto částem obrazu. Poté se pro všechny tyto oblasti spočítá histogram a na základě něj se pak v obrazu hledají oblasti, které jsou tomuto histogramu nejpodobnější. Poté se vybere pouze oblast, která je pro danou problematiku důležitá (silnice) a na hranice této oblasti se aproximuje přímka, která odpovídá silnici. Tato metoda není vhodná pro použití do městského prostředí, jelikož tam je obraz různorodější a skládá se mnohdy z více než tří částí. O tomto způsobu detekce pojednává publikace [11]. Příkladem tohoto způsobu je obrázek 4.

2.2 Detekce s učením

Díky zvyšování výpočetního výkonu a rozmachu umělé inteligence se v dnešní době využívá k detekci jízdných pruhů i strojového učení, konkrétněji hlubokého učení.

(a) Pomocí kovnolučních sítí lze detekovat i samotné jízdní pruhy.[5]

(b) Příklad výstupu z konvoluční neuronové sítě pro detekci pruhů. Tento model signalizuje oblast jízdního pruhu pomocí zeleně vykreslené oblasti.[1]

Obrázek 5: Ukázka detekce pruhů pomocí hlubokého učení

Program s učícím se algoritmem má většinou dvě fáze. Trénování a normální běh. Pro trénování je potřeba získat vstupní data, podle kterých bude model natrénován a data, která chceme, aby náš model dokázal rozpoznat. Pokud například bude úkolem vytvořit model, který dokáže rozpoznat, zda je na obrázku auto nebo motorka, tak je potřeba si sehnat obrázky aut a motorek a ke každému obrázku napsat, co na obrázku je.

Trénování neuronové sítě je založeno na algoritmu zpětné propagace [6]. Neuronová síť dostane na vstupu data. Tyto data se sítí vyhodnotí a v poslední vrstvě se zjistí, jak moc se síť spletla. Tímto se vypočítá chyba modelu, která následně putuje zpátky sítí a upravuje váhy modelu.

Z principu toho jak neuronové sítě fungují se dá jednoduše poznat, že při trénování je potřeba hodně výkonu. Normální běh už natrénovaného modelu spočívá pouze ve vyhodnocení dat sítí, což nevyžaduje tolik výkonu, takže natrénovaný model může běžet i na slabším stroji.

Modely které se dají použít pro detekci jízdních pruhů se nazývají YOLO[13] nebo Faster RCNN[14]. Tyto modely využívají hlubokých konvolučních neuronových sítí. V praxi jsou už tyto modely natrénovány na určitém obecném datasetu. Pro použití na specifický problém se tyto modely nemusí trénovat úplně od nuly, ale natrénuje se pouze pár posledních vrstev. Tyto modely mají velkou výhodu v tom, že dokáží detekovat více objektů najednou. Dokážeme tedy jedním modelem rozpoznávat jak jízdní pruhy, tak navíc i auta nebo chodce. Příklad využití učícího algoritmu pro takový problém je v publikaci [4]. Na obrázku 5 jsou vidět dva způsoby jak pomocí umělé inteligence detekovat jízdní pruhy.

3 Předzpracování obrazu pro detekci čar na vozovce

Algoritmy, které nejsou založené na strojovém učení, většinou obsahují několik dílčích kroků, které jsou aplikovány na vstupní obraz. Tyto kroky a operace spadají do kategorie předzpracování obrazu. Při tomto procesu se vylepšuje kvalita obrazu za účelem zvýšení pravděpodobnosti správné detekce oblastí, které jsou pro danou problematiku důležité. Nejčastějšími metodami předzpracování jsou filtrování šumu, zostření obrazu, převod na stupně šedi, binarizace nebo korekce barev a jasu. Použití různých operací předzpracování závisí na konkrétním problému.

3.1 Vyhlazení obrazu

Vyhlazení obrazu slouží pro potlačení šumu, který vznikl při digitalizaci obrazu. Existuje více způsobů, kterým tohoto dosáhneme. V této podkapitole popíšu různé metody vyhlazení obrazu, které se dají použít. Na obrázku 6(a) je k vidění původní obraz, na kterém budu testovat různé druhy filtrů.

Průměrování

Jedním z možných filtrů pro vyhlazení obrazu je průměrování. Princip tohoto vyhlazování je jednoduchý. Pro každý pixel v obraze se spočítá průměrná hodnota se všemi sousedy. Touto hodnotou se daný pixel nahradí. Největší problém tohoto filtru je, že pokud se v obrazu vyskytne pixel, který je velice odlišný od původní hodnoty, dokáže velmi negativně ovlivnit své sousedy. Dalším problémem je, pokud tento filtr narazí na hranu. Po aplikování takového filtru hrana nebude tak ostrá, což může ovlivnit další zpracování obrazu. Aplikace takového filtru o velikosti 5x5 je vidět na obrázku 6(b)

Mediánový filtr

Tento filtr postupně prochází všechny pixely v obraze a kontroluje i všechny jeho sousední pixely. Ze sousedních pixelů vybere medián (prostřední hodnotu) a touto hodnotou nahradí daný pixel. Tento filtr je velice účinný proti šumu, který se v anglické literatuře nazývá „salt-and-pepper noise“. Méně účinný je proti Gaussovskému šumu, kde kvalita obrazu po aplikování filtru již není tak dobrá. Příklad tohoto filtru o velikosti oblasti 5x5 je na obrázku 6(c)

Gaussův filtr

Gaussův filtr je speciální operátor pro potlačení šumu. Podle názvu lze snadno předpokládat, že tento filtr nejlépe odstraňuje Gaussův šum. Přesný vzorec pro vypočítání Gaussovy funkce je:

$$G(x,y) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{\frac{-x^2+y^2}{2\sigma^2}} \tag{1}$$

V praxi se však tento vzorec aproximuje do matice. Například pro matici 5x5 se směrodatnou odchylkou 1 vypadá matice takto:

$$G = \frac{1}{273}\begin{vmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{vmatrix} \tag{2}$$

Tuto matici pak musíme vydělit součtem všech elementů uvnitř matice, čímž dostaneme Gaussův filtr. Na obrázku 6(d) lze vidět použití Gaussova filtru o velikosti 5x5. Lze si všimnout, že není tak agresivní jako metoda průměrování a soustředí se hlavně na eliminaci šumu.

Bilaterální filtr

Tento poněkud sofistikovanější filtr vyhlazuje nejen na základě váženého průměru okolních pixelů, ale také na základě podobnosti barvy. Výsledkem tohoto filtru je vyhlazený obraz, který má však stále ostré hrany. Tato metoda má několik parametrů. Klasickým parametrem je velikost matice. Druhým parametrem je práh podobnosti barvy. Pokud je tento parametr nastaven na vysokou hodnotu, obraz bude mít málo barev a bude vypadat trochu jako z animovaného filmu. Příklad bilaterálního filtru o velikosti 7x7 a prahu oblasti barvy 90 je na obrázku 6(e)

3.2 Převod na stupně šedi

Dalším krokem po vyhlazení obrazu je převod barev na stupně šedi. Důvodem této úpravy je zejména urychlení zpracování informací, protože ze třech barevných kanálů zůstane jen jeden, takže výsledkem je 3x méně dat, které je třeba zpracovat. Převod na stupně šedi se dá použít tam, kde barva obrazu není klíčovou informací, tudíž ji lze zanedbat (například detekce všech hran v obraze). Při převodu obrazu na stupně šedi se prochází všechny pixely v obrazu a aplikuje se na ně následující vzorec:

$$Y = 0.299R + 0.587G + 0.114B \tag{3}$$

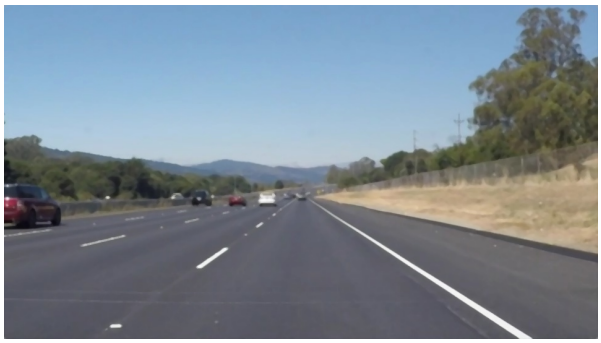
, kde proměnné R, G a B odpovídají hodnotám jednotlivých barevných kanálů červené, zelené a modré.

3.3 Binarizace obrazu

Binarizace obrazu umožňuje jednoduchou segmentaci obrazu na jednotlivé objekty a pozadí. Tento proces se dá aplikovat jednoduše na černobílé obrazy a trochu komplikovaněji i na barevné obrazy.

(a) Původní obraz

(b) Obraz filtrovaný průměrováním

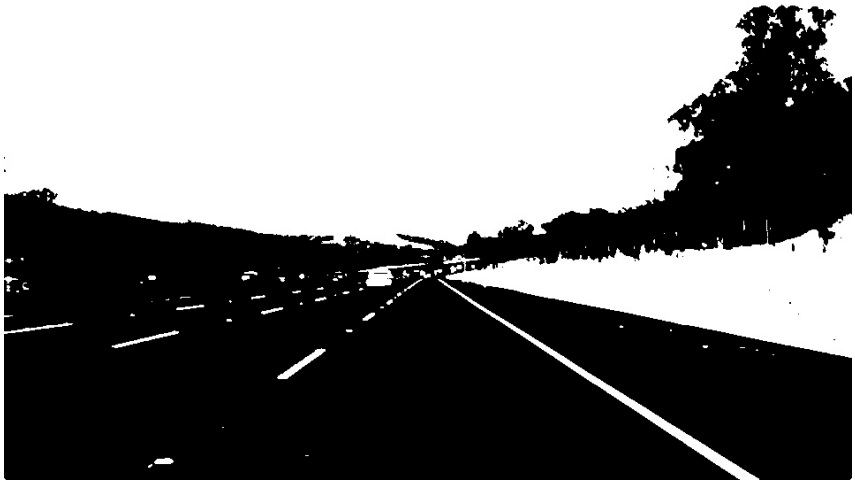


(c) Obraz filtrovaný mediánovým filtrem

(d) Obraz filtrovaný Gaussovým filtrem

(e) Obraz filtrovaný bilaterálním filtrem

Obrázek 6: Srovnání filtrů



Obrázek 7: Binární obraz

(a) Původní obraz

(b) Obraz rozmazaný Gaussovým filtrem o velikosti 21x21

(c) Výsledný zostřený obraz

Obrázek 8: Ukázka metody unsharp masking

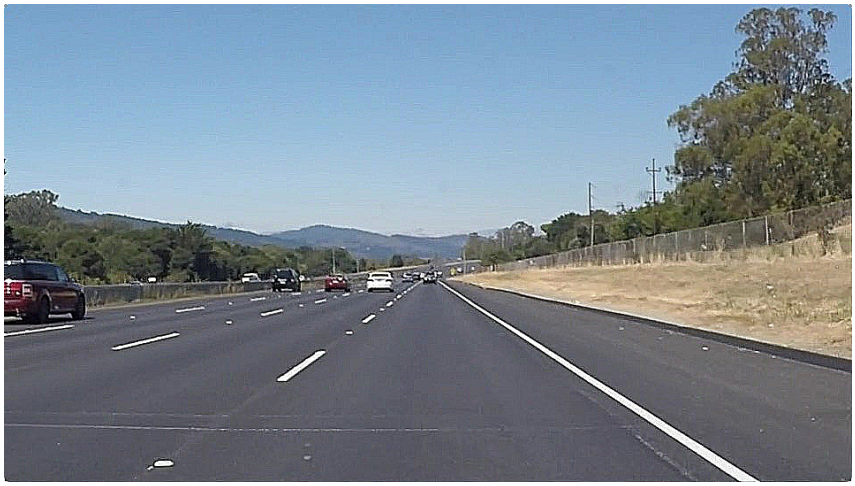
Princip funkce je takový, že na začátku je určena hranice hodnoty pixelu, která bude rozhodovat, zda bude pixel bílý nebo černý. Pak se projde celý obraz a hodnota každého pixelu se porovná s touto hranicí. Pokud je hodnota větší než daná hranice, pixel bude bílý. Pokud je hodnota menší, pixel bude černý. Je možné dané barvy i obrátit, to znamená, že pokud bude hodnota pixelu větší než hranice, pixel bude černý. Na obrázku 7 můžeme vidět, že při binariazci jsou dobře vidět čáry jízdních pruhů díky kontrastu barvy silnice a silniční čáry.

3.4 Zostření obrazu

Zostření obrazu je proces opačný k vyhlazování, které jsem popsal dříve. Používá se tehdy, pokud je nutno z obrazu vytáhnout více detailů. Pozitivním následkem této operace jsou ostřejší hrany. Nevýhoda je ve zvýšeném množství šumu, které zostřením vznikne.

Unsharp masking

Jedna z metod jak zostřit obraz se anglicky jmenuje „unsharp masking“. Tato metoda funguje tak, že z původního obrazu je vytvořen další obraz, který je vyhlazen Gaussovým filtrem. Z



Obrázek 9: Zostřený obraz pomocí konvoluce. Lze pozorovat zvýšený výskyt šumu.

tohoto vyhlazeného obrazu se udělá negativ a sečte se s původním obrazem. Proces této metody je ilustrován na obrázku 8.

Zostření pomocí konvoluce

Další běžnou metodou zostřování obrazu je zostření pomocí konvoluce. Tento způsob funguje stejně jako při rozmazávání obrazu například pomocí Gaussova filtru. Také se vytvoří matice, která je posouvána po obrázku, avšak hodnoty v matici jsou jiné. U tohoto postupu se používá takzvaný Laplacián. U Gaussova filtru jsou všechny hodnoty kladné. Naproti tomu u Laplaciánu je kladná jen hodnota pixelu ve středu matice. Ostatní pixely mají negativní koeficient. Příkladem takové matice je:

$$\begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix} \quad nebo \quad \begin{vmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{vmatrix} \tag{4}$$

Výsledkem aplikování jedné z matic je zostřený obraz. Pokud je použita ta vlevo, obraz bude zostřený méně, než když bude aplikována ta vpravo. Na obrázku 9 je vidět použití konvoluce se silnější maticí pro zostření obrazu.

4 Detekce hran při detekci pruhů

Detekce hran je klíčovou operací při zpracování obrazu. Umožňuje detekovat různé objekty nebo jejich části. V této kapitole nejdříve vysvětlím co to vlastně hrana v obrazu je. Poté popíšu existující operátory pro detekci hran a nakonec provedu porovnání výsledků těchto detektorů.

4.1 Charakteristika hrany

V obrazu je hrana reprezentována jako pixely, kde dochází ke změně jasu nebo barvy. Hrana v obraze je dána vlastností obrazu a jeho okolí. V případě mého úkolu je hrana například přechod mezi povrchem vozovky a čárou na vozovce nebo přechod mezi silnicí a vozidlem. Na obrázku 10(a) je obraz, na kterém budu testovat různé detektory hran.

4.2 Sobelův operátor

Sobelův operátor provádí aproximaci gradientu v obrazu, čímž umožňuje detekovat hrany v černobílém obrazu. Je to dvojice 3x3 matic, které jsou vzájemně otočené o 90 stupňů. Obě matice jsou navrženy tak, aby co nejvíce reagovaly na hrany. Jedna matice je pro horizontální a druhá matice je pro vertikální směr hran.

$$Gx = \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}, Gy = \begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix} \tag{5}$$

Tyto dvě matice mohou být použity jak zvlášť, pro zjištění hran v daném směru, nebo i dohromady pro zjištění absolutní hodnoty gradientu v každém bodě obrazu. Pro výpočet absolutní hodnoty gradientu se pak používá vzorec:

$$G = \sqrt{Gx^2 + Gy^2} \tag{6}$$

Pro rychlejší výpočet se však v praxi tento výpočet aproximuje na:

$$G = |Gx| + |Gy| \tag{7}$$

Další užitečnou věcí při použití Sobelova operátoru je určení směru gradientu pomocí následujícího vzorce. S touto informací se dále pracuje například v Cannyho detektoru. Příklad Sobelova detektoru můžete vidět na obrázku 10(b).

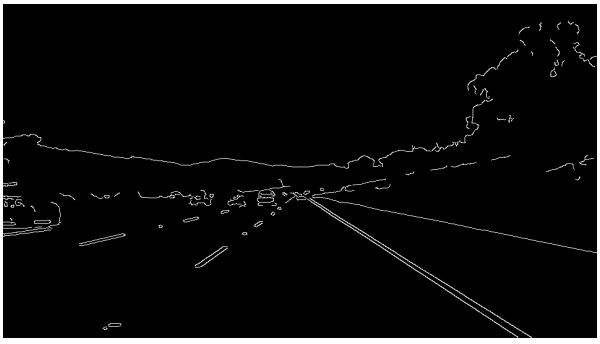
$$\theta = atan2(\frac{Gy}{Gx}) \tag{8}$$

(a) Původní obraz

(b) Příklad Sobelova operátoru

(c) Příklad Prewittova operátoru

(d) Příklad Laplaciánu



(e) Příklad Cannyho detektoru

Obrázek 10: Srovnání detektorů hran

4.3 Prewittův operátor

Prewittův operátor se také jako Sobelův operátor skládá ze dvou matic o velikosti 3x3. Hodnoty v matici jsou však jiné. Na rozdíl od Sobelova operátoru je jeho implementace o něco jednodušší, avšak je více citlivý na šum. Tento operátor se také nejčastěji aplikuje na černobílé obrazy. Stejně jako u Sobelova operátoru tyto dvě matice můžeme použít jak zvlášť, tak dohromady a můžeme jejich výsledek použít k výpočtu směru gradientu. Použití Prewittova operátoru je vidět na obrázku 10(c)

$$Gx = \begin{vmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{vmatrix}, Gy = \begin{vmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{vmatrix} \tag{9}$$

4.4 Laplaceův operátor

Dalším možným operátorem pro detekci hran je Laplaceův operátor. Na rozdíl od dvou předchozích operátorů je Laplacián pouze jedna matice, která detekuje zároveň hrany ve směru x a y. Tato matice má tvar:

$$G = \begin{vmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{vmatrix} \tag{10}$$

Existuje také varianta, která detekuje i diagonální hrany. Ta je ve tvaru:

$$G = \begin{vmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{vmatrix} \tag{11}$$

V této práci už jsem o Laplaciánu psal v souvislosti se zostřováním obrazu. Matice má stejný tvar a má i skoro stejné hodnoty. Pouze hodnota ve středu matice je u zostřování obrazu o jedničku vyšší. Použití Laplaceova operátoru je k vidění na obrázku 10(d)

4.5 Tan-Triggsova metoda

Tento přístup není detektorem hran v pravém smyslu slova. Tato metoda je určena hlavně pro extrakci příznaků tváře v obrazu se špatnými světelnými podmínkami. Pojednává o ní publikace [12]. Tento přístup nejprve provede filtraci obrazu Gaussovými filtry s různými velikostmi matice a s různými směrodatnými odchylkami. Poté se tyto dva obrazy od sebe odečtou. Nakonec se provede vyrovnání kontrastu, jehož výsledkem jsou výrazné rysy daného obrazu.

I přesto že tato metoda není určena pro detekci hran, tak jsem se rozhodl tento přístup vyzkoušet pro detekci jízdního pruhu ve špatně osvětleném prostředí. Mými experimenty jsem však došel k tomu, že v normálních světelných podmínkách se tato metoda dá zhruba použít

(a) Původní obraz za denního světla

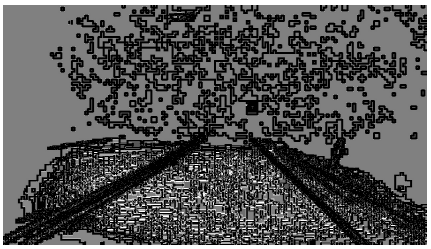
(b) Příznaky z metody Tan-Triggs

(c) Hrany z metody Tan-Triggs

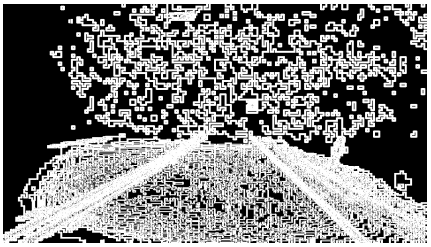
Obrázek 11: Metoda Tan-Triggs za denního světla

pro detekci čar, avšak v nočním prostředí je tato metoda pro detekci jízdních pruhů v podstatě nepoužitelná. Tyto experimenty jsou zobrazeny na obrázcích 11 a 12.

(a) Původní obraz za nočního světla



(b) Příznaky z metody Tan-Triggs



(c) Hrany z metody Tan-Triggs

Obrázek 12: Metoda Tan-Triggs v noci

4.6 Cannyho detektor

Cannyho detektor je další způsob detekování hran v obraze. Obecné vlastnosti tohoto detektoru jsou nízká chybovost, přesná lokalizace středu hrany a nízký počet falešných hran. Na rozdíl od jednoduchých operátorů je Cannyho detektor složen z více kroku, který posouvá možnosti detekce hran zase o kus dále. Efekt Cannyho detektoru ilustruje obrázek 10(e)

Aplikace Gaussova filtru a určení gradientu

Nejprve se na obraz na vstupu aplikuje Gaussův filtr. Tímto krokem se obraz vyhladí a odstraní se výskyt šumu. Poté se použije jeden z operátorů pro detekci hran popsaných výše. Tímto dostaneme obraz, který zvýrazňuje hrany v obraze, které chceme detekovat.

Nalezení lokálních maxim

Tato metoda slouží ke "ztenčení"hrany. Jako hrana se berou body, které jsou lokálními maximy. Odeberou se tedy body, které mají nižší hodnotu gradientu a to tak, že se porovnává hodnota gradientu se sousedy ve směru a proti směru gradientu, který byl zjištěn za použití operátoru pro detekci hran. Pokud je tato intenzita vyšší, bod se stane součástí hrany. Pokud ne, tato hodnota je potlačena.

Prahování s hysterezí

Posledním krokem Cannyho detektoru je rozhodování, co se nakonec bere jako hrana a co ne. Nejprve jsou zvoleny dvě hodnoty prahu. Jedna hodnota určuje minimální velikost prahu a druhá hodnota určuje maximální velikost prahu. Pokud je intenzita bodu pod minimálním prahem, určitě to není hrana. Naopak jestliže je intenzita bodu nad maximálním prahem, jistě to hrana je. Pro ostatní body, které se nachází mezi těmito prahy pak záleží, jestli je tento bod spojen s bodem, který je určitě hranou. Pokud ano, tento bod se také bere jako hrana. Pokud ne, bod hranou není.

5 Detekce přímek Houghovou transformací

Aby bylo možné detekovat jízdní pruh, tak je vhodné nejprve detekovat postranní čáry. Nejjednodušším případem je rovinka, kdy jsou postranní čáry jednoduché přímky.

Pro detekci přímek využiji výstup z detektoru hran, který určí body v obraze, kde jsou hrany. Z těchto bodů pak dokážu určit, zda tyto body tvoří přímku. Začnu obecnou rovnicí přímky, která vypadá takto:

$$y = ax + b \tag{12}$$

Pomocí této rovnice lze přímku zakreslit do roviny x,y. V této chvíli jsou x,y proměnné a a,b parametry. Toto však lze změnit a upravit rovnici na:

$$b = -ax + y \tag{13}$$

Tímto se dají body znázornit přímkou v rovině a,b. Pokud se dvě přímky protnou v bodě a a b, znamená to, že leží na stejné přímce, s parametry a, b. Tento přístup funguje pro všechny možné přímky až na jednu výjimku a tou je situace, kdy je přímka rovnoběžná s osou Y.

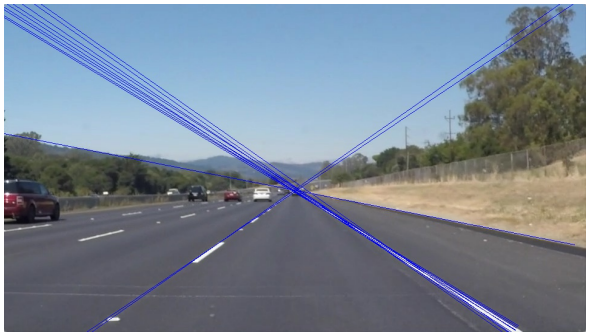
Cestou ven z tohoto problému je převod na polární souřadnice, kde rovnice přímky vypadá následovně:

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta \tag{14}$$

V tomto vzorci θ vyjadřuje úhel přímky od počátku v radiánech v intervalu <-90°, +90°> a ρ vyjadřuje vzdálenost přímky od počátku soustavy v pixelech.

Tímto vzorcem se spočítá rovnice pro všechny hranové body a určí se hranice, kolik bodů musí být na jedné přímce, aby se dala opravdu považovat za přímku.

Pokud v praxi je tato hranice nízká, bude se v obrazu detekovat obrovské množství přímek. Pokud se však zvýší, zůstanou opravdu jen přímky, které odpovídají místům největší intenzity hranových bodů.



(a) Klasická Houghova tranformace

(b) Pravděpodobnostní verze Houghovy transformace

Obrázek 13: Srovnání klasické a pravděpodobnostní verze Houghovy transformace

5.1 Pravděpodobnostní Houghova transformace

Postup který jsem právě popsal v teorii funguje výborně. Pokud je však v praxi třeba detekované přímky přesně vykreslit tak nastává problém, jelikož klasická Houghova transformace vrací pouze parametry přímky. Avšak například v mé práci potřebuji spíše koncové body detekované přímky, abych detekovanou přímku dokázal přesně vykreslit. S klasickou Houghovou transformací jsem schopen tyto koncové body dopočítat, ale pouze takovým způsobem, že přímka bude vést přes celý obraz, což je zrovna v mém případě naprosto nevhodné.

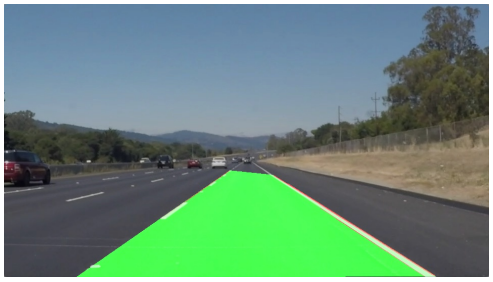
Ve svém programu používám funkci knihovny OpenCV s názvem *HoughLinesP*. Tato funkce odpovídá pravděpodobnostní verzi Houghovy transformace, se kterou přišel tým ve kterém byli i dva čeští výzkumníci.[7] Tento algoritmus nebere v potaz všechny možné body na přímce, ale pouze náhodné podmnožiny bodů, které postačují k vytvoření přímky, což výrazně snižuje výpočetní náročnost tohoto algoritmu. Další výhodou tohoto řešení je to, že vrací přímku ve tvaru dvou hraničních bodů, takže lze přesně vykreslit detekovanou přímku. Srovnání klasické a pravděpodobnostní verze Houghovy transformace je k vidění na obrázku 13.

6 Vykreslení oblasti jízdního pruhu

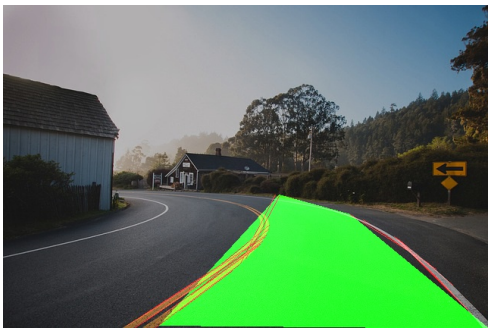
Vykreslením přímek do obrazu bych svůj úkol detekce jízdního pruhu dostatečně vyřešil. Chtěl jsem však své řešení dostat ještě o krok dále a vykreslit oblast silnice, kterou silniční čáry vymezují. Je to také z toho důvodu, že samotné přímky v obraze jdou lehce přehlédnout.

Ve vykreslování této oblasti jsem se nikde neinspiroval a hledal jsem od začátku své vlastní řešení. První nápad, který jsem dostal, byl následující. Z detekovaných přímek jsem si zjistil nejbližší a nejvzdálenější body levých a pravých přímek od kamery. Z těchto čtyř bodů jsem poté sestrojil čtyřúhelník, který měl odpovídat jízdnímu pruhu. Toto řešení fungovalo správně pouze na rovných cestách. V zatáčkách tato oblast většinou zasahovala do protijedoucího pruhu. Tento způsob je zobrazen na obrázku 14.

Využil jsem tedy toho, že zatáčky jsou detekovány větším počtem kratších přímek, jejichž koncové body v podstatě kopírují zakřivení silniční čáry. Nejprve tedy všechny přímky prodloužím směrem ke kameře, aby začínaly mimo obraz. Poté si zjistím koncové body všech přímek, uložím si je do pole a seřadím podle hodnoty souřadnice X. Poté na začátek pole přidám pozici bodu levé čáry, která je nejblíže kamery a na konec pole přidám pozici bodu pravé čáry, která je nejblíže kamery. Pozice těchto bodů jsou znázorněny na obrázku 15. Tímto se vytvoří polygon, který začíná u levé čáry a kopíruje její tvar. U posledního koncového bodu levé přímky překříží jízdní pruh na první koncový bod pravé přímky a pokračuje až ke koncovému bodu pravé přímky, který je nejblíže kamery. Pro ilustraci je přiložen obrázek 16.



(a) Rovná cesta



(b) Zatáčka

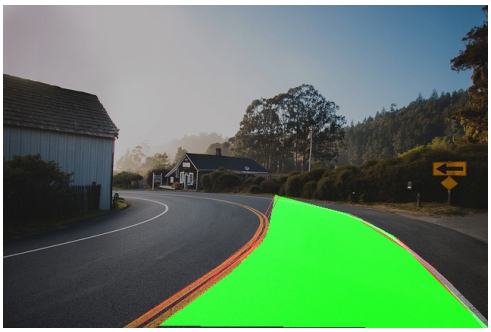
Obrázek 14: Použití první metody vykreslení oblasti jízdního pruhu

(a) Body na rovné cestě

(b) Body v zatáčca

Obrázek 15: Znázornění řídících bodů polygonu pro vykreslení oblasti jízdního pruhu

(a) Nová metoda vykreslení segmentu na rovince

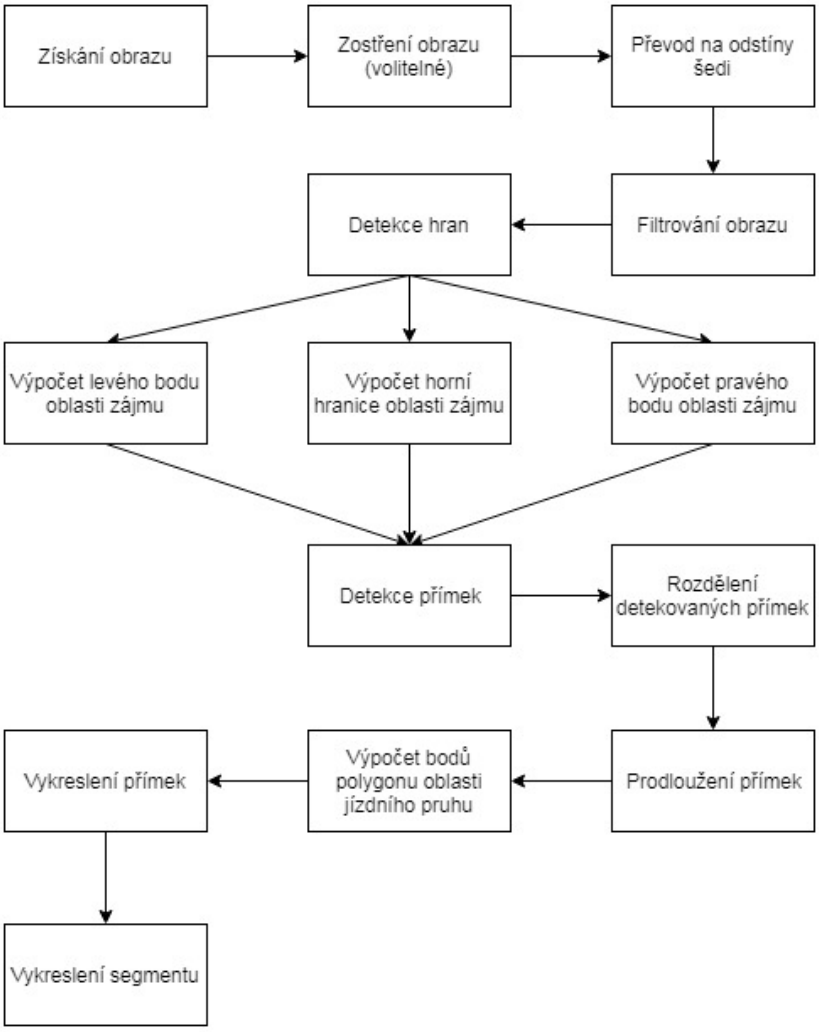


(b) Nová metoda vykreslení segmentu v zatáčce

Obrázek 16: Použití nové metody vykreslení oblasti jízdního pruhu

7 Implementace řešení

V této kapitole budu popisovat postup vyřešení mého úkolu. Nejdříve zde popíšu jaké nástroje pro řešení mého úkolu jsou dostupné, jaké nástroje byly při vytváření nakonec použity a následně budu vysvětlovat jak jsem postupovat při detekci čar na vozovce. Tento postup je znázorněn i formou vývojového diagramu na obrázku 17.



Obrázek 17: Vývojový diagram mého řešení

7.1 Knihovny pro zpracování obrazu

V dnešní době už existuje více knihoven, které poskytují základní i pokročilé nástroje potřebné pro zpracování obrazu. V této kapitole popíšu dvě z knihoven, které se používají pro zpracování obrazu. Obě dvě tyto knihovny jsou i nadále aktivně vyvíjeny, takže se stále pracuje na jejich vylepšení a přidávání nových funkcí.

OpenCV

Jednou z knihoven pro zpracování obrazu je knihovna OpenCV[15]. V době psaní této práce je nejnovější verze této knihovny 3.4.1.

Tato knihovna cílí hlavně na multiplatformnost, tudíž byla napsána v jazyce C. Od verze 2.0 ovšem nabízí i rozhraní pro další jazyky jako například C++, Java nebo Python. Tato knihovna běží jak pod operačním systémem Linux, tak Windows, dokonce i pod MacOS. Existuje i verze této knihovny pro Android, takže ji lze využít i v mobilních aplikacích. V roce 2010 byla do OpenCV přidána i podpora technologie CUDA.

OpenCV zahrnuje širokou škálu funkcí. Poskytuje základní funkce pro zpracování obrazu, jako například transformace, detekci hran, nebo i funkce pro předzpracování obrazu (Gaussův filtr nebo mediánový filtr)

Nabízí také základní nástroje strojového učení a sice Support vector machine a k-nearest neighbors. Obsahuje také základní neuronové sítě jako například MLP. Dalšími nástroji pro strojové učení v OpenCV jsou například statistické modely nebo rozhodovací stromy.

Součástí této knihovny jsou i nástroje pro zpracování videa jako je například detekce pohybu a sledování objektů.

V neposlední řadě tato knihovna obsahuje i nástroje pro spojování obrazu, nástroje pro 3D rekonstrukci obrazu nebo i řešení pro detekci objektů jako je například Haarova kaskáda, nebo speciální verze SVM algoritmu.

Dlib

Druhou poměrně známou knihovnou pro zpracování obrazu je Dli[16]. Nejnovější stabilní verze k aktuálnímu datu je 19.10.

Narozdíl od OpenCV je tato knihovna napsána přímo v jazyce C++. Podobně jako OpenCV i Dlib obsahuje rozhraní pro jiné programovací jazyky, avšak obsahuje pouze rozhraní pro Python.

Stejně jako OpenCV tato knihovna samozřejmě poskytuje základní funkčnosti ohledně zpracování obrazu. Obsahuje velké množství filtrů, detektorů nebo transformací. V oblasti detektorů poskytuje oproti OpenCV například i SURF technologie nebo HOG.

Oproti OpenCV však Dlib nabízí větší množství nástrojů pro strojové učení. Jsou to algoritmy jako MLP, široká škála SVM algoritmů, ale v této knihovně jsou přítomny i nástroje na deep learning. Obsahuje nástroje na binární klasifikaci, regresi, predikci, ale také konvoluční neuronové sítě nebo učení bez učitele.

Mezi další funkčnost knihovny Dlib patří nástroje pro práci s grafy nebo lineární algebrou. Obsahuje i technologie pro komunikaci po síti, nebo algoritmy pro optimalizace. Součástí této knihovny jsou i funkce pro shlukování dat, transformaci dat a obecně nástroje pro datovou analýzu.

7.2 Použité nástroje

Mým úkolem bylo vytvořit aplikaci, která v obraze dokáže detekovat jízdní pruhy. Před tím, než jsem tento úkol začal řešit, tak jsem se rozhodoval, jaký jazyk použiji. V úvahu přicházely jazyky C++ a Python. Nakonec jsem se přiklonil k Pythonu. Sice se rychlostí nemůže vyrovnat jazyku C++, ale mám s Pythonem větší zkušenosti a také má uživatelsky přívětivé datové struktury. Výhodou také je, že většina knihoven pro zpracování obrazu obsahuje wrapper pro Python.

Při výběru knihovny pro daný úkol jsem sáhnul po OpenCV. Jednak už mám s touto knihovnou určitě zkušenosti (jak v C++, tak v Pythonu) a dále nepoužívám v tomto projektu strojové učení, takže mi stačí základní nástroje pro zpracování obrazu, které OpenCV poskytuje. Společně s knihovnou Numpy jsem byl schopen vyřešit do míry, kdy jsem s mým řešením poměrně spokojen.

7.3 Použité postupy

Můj program obsahuje volitelné přepínače, kterými lze zvolit metodu zostření obrazu, filtr pro vyhlazení obrazu a případně i detektor hran. Jediným povinným parametrem je vstupní obraz, kterým může být jak video, tak i jeden soubor s obrazem.

Pokud je zvolena některá z metod zostření obrazu, tak se jako první krok provede právě tato úprava. Pokud tento přepínač není použit, pokračuje se dalším krokem.

V dalším kroku se celý obraz převede na černobílý, protože nepotřebuji informaci o barvě pruhu, jelikož pruhy jsou většinou bílé, občas i žluté. Tyto barvy jsou však stále v kontrastu s tmavou vozovkou, takže hrany půjdou dobře vidět i z černobílého obrazu.

Poté se na celý obraz aplikuje filtr kvůli snížení šumu. Pokud nebyl zadán přepínač pro zvolení filtru, použije se gaussův filtr. Dalším krokem je detekce hran. Pokud ani zde nebyl zvolen žádný přepínač pro detektor hran, je implicitně použit Cannyho detektor.

Poté se provede selekce oblasti, která nejpravděpodobněji obsahuje celou silnici. Tato oblast je dána čtyřmi body. První dva body jsou levý spodní roh a pravý spodní roh. Zbývající dva body se poté dopočítají podle velikosti obrázku. Tento krok pomůže eliminovat ostatní detekované hrany, které nijak nesouvisí s jízdním pruhem.

Následně aplikuji pravděpodobnostní Houghovu transformaci pro detekci přímek, která detekuje silniční pruhy. Tento algoritmus vrátí koncové body všech přímek, které detekoval.

V dalším kroku si rozdělím přímky na levé a pravé podle jejich sklonu. Toto rozdělení je důležité kvůli tomu, v jakém pořadí vrací hraniční body přímek Houghova transformace. Každá přímka, kterou vrátí Houghova transformace je pole, které je složeno ze dvou bodů. První bod v tomto poli je vždy bod s nižší hodnotou souřadnice X. První bod levých přímek je tedy ten spodní, kdežto první bod pravých přímek je ten vrchní. Při tomto rozdělení také ignoruji přímky, které jsou horizontální, jelikož tyto přímky neodpovídají dopravnímu značení. Toto je ošetřeno tak, že obě souřadnice na ose Y dané čáry od sebe musí být vzdáleny více jak 40 pixelů, což zamezí vykreslení horizontálních čar.

Poté si pro jistotu všechny přímky protáhnu směrem ke kameře. To dělám kvůli tomu, že někdy jsou čáry přerušované a to zejména ty levé. Pokud bych přímky neprotáhl, mohlo by se stát, že by některá z čar nezačínala na spodní straně obrazu, ale výrazně výše.

Z prodloužených přímek se pak vypočítají body polygonu, který odpovídá oblasti jízdního pruhu na silnici. Pak se tento polygon vyplní zelenou barvou a vykreslí se i detekované přímky.

8 Testování

Po napsání základní funkčnosti programu jsem musel experimentálně vyzkoušet několik věcí, abych zajistil co nejoptimálnější a nejuniverzálnější funkčnost na různých vstupních souborech. Zdroje obrázků na kterých byly obrázky testovány jsou: [17], [18], [19], [20], [21]

8.1 Selekce oblasti

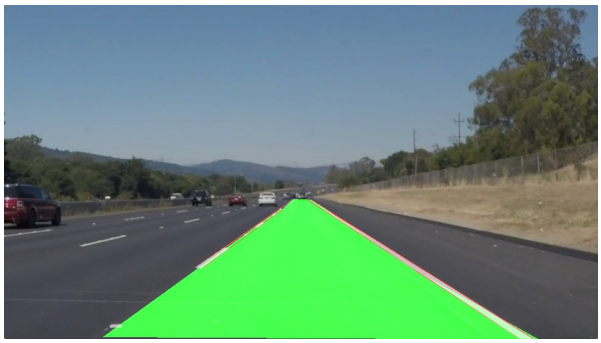
Nejprve jsem musel experimentálně zjistit, kde v obraze se nejpravděpodobněji vyskytuje co největší část silnice pro detekci čar. Osvědčil se mi lichoběžník, jehož spodní strana je na spodní straně obrazu a zabírá celou šířku obrazu. Horní strana lichoběžníku je přibližně ve 42 procentech výšky obrazu. Tato vrchní hranice by měla být ideálně v místě úběžníku daného obrazu. To je bod, kde se silniční čáry protínají. Dále si spočítám horizontální střed obrazu a horní dva body lichoběžníku jsou 30 bodů nalevo a 150 bodů napravo od tohoto středu. Pro lepší pochopení je přiložen obrázek 18.

Obrázek 18: Selekce oblasti

8.2 Testování hotového řešení

Obrázky v této kapitole ukazují testování mého řešení v různých situacích. Program dokáže snadno detekovat jízdní pruhy na rovných cestách a dokáže rozpoznat jízdní pruh i v mírných zatáčkách. Program byl také otestován na videu. Příklady použití v různých prostředích může být viděno na obrázku 19.

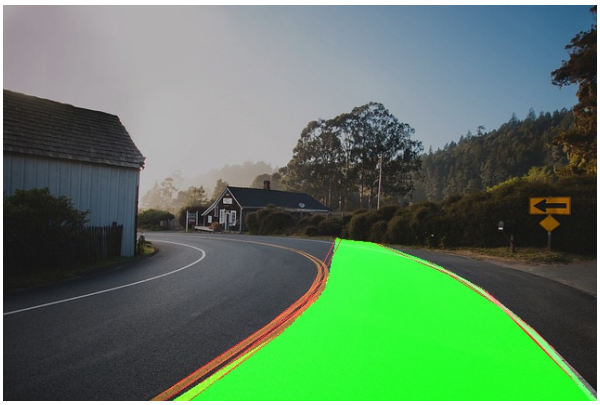
Program byl testován i v nočním prostředí. Použití mého programu v noci silně závisí na kvalitě osvětlení vozidla. Výhodou nočního obrazu je takřka úplná eliminace okolních hran,



(a) Rovná cesta

(b) Jiná rovnika

(c) Zatáčka



(d) Jiná mírná zatáčka

Obrázek 19: Ukázka použití mého řešení

Obrázek 20: Detekce jízdního pruhu v noci

jelikož světla jsou zaměřena přímo na cestu. Avšak toto ovlivňuje i kvalitu jízdních pruhů, které v noci jsou vidět pouze na krátkou vzdálenost. Obrázek 20 ilustruje použití programu v noci.

8.3 Analýza doby běhu programu

Při testování jsem také prováděl analýzu doby běhu mého řešení. Toto testování bylo prováděno za účelem zjištění, která část programu trvá nejdéle. Testování probíhalo na obrázku 17(d), který má rozměry 426 x 640 pixelů. Měření doby běhu probíhalo na laptopu Lenovo E550. Tento počítač má dvoujádrový procesor Intel i3 4005U s taktem 1,7 GHz. Dále má operační paměť typu DDR3 o velikosti 4GB o frekvenci 800 MHz, a integrovanou grafickou kartu. Výsledky tohoto měření jsou v tabulce 1. Poslední řádek tabulky je čas běhu programu bez použití zostření, s použitím Gaussova filtru, Cannyho detektoru a zobrazením výsledku.

	Doba běhu (ms)
Zostření pomocí konvoluce (slabší matice)	4
Zostření pomocí konvoluce (silnější matice)	7
Zostření pomocí odečtení obrazu	8
Převod na stupně šedi	4
Aplikace Gaussova filtru o velikosti 7x7	4
Aplikace průměrování o velikosti 7x7	7
Aplikace mediánového filtru o velikosti 7x7	45
Aplikace bilaterálního filtru o velikosti 7x7	49
Cannyho detektor	8
Sobelův detektor (oba směry)	8
Laplacián	2
Prewittův detektor (oba směry)	6
Selekce oblasti a detekce přímek	40
Rozdělení přímek	1
Vykreslení oblasti jízdního pruhu	16
Vykreslení přímek	12
Celkový čas běhu programu	98

Tabulka 1: Analýza běhu částí programu

Z tabulky lze vidět, že nejdéle běží část se selekcí oblasti a detekci přímek. Z filtrů pro vyhlazení obrazu nejhůře dopadl mediánový a bilaterální filtr, které by tudíž nemohly být použity pro zpracování obrazu v reálném čase, jelikož by prodloužili běh programu o celou jednu polovinu celkového času.

9 Závěr

Detekce jízdních pruhů je problém, který v dnešní době jde už poměrně dobře řešit. Mnou navržené řešení dokáže snadno rozpoznat jízdní pruhy na rovné cestě a poradí si také s mírnými zatáčkami. Řešení je nezávislé na druhu silničního značení, což znamená, že se dá použít jak v Evropě, kde jsou čáry bílé a místy mohou být středové čáry přerušované, tak i v jiných zemích (například USA), kde mají silniční čáry bílé a žluté.

Mnou navržené řešení by se dalo ještě vylepšit o funkci aproximace křivky, aby byla detekce v zatáčkách přesnější a robustnější. Nabízí se také použití umělé inteligence, která by přesnost velkou měrou vylepšila. S dnešními knihovnami a potřebnými znalostmi není problém takový program realizovat.

Podle provedeného měření rychlosti běhu je vidět, že zpracování jednoho obrazu trvá skoro 100 milisekund, což odpovídá snímkovací frekvenci 10 FPS. Tato frekvence je pro použití mého programu v reálném čase poměrně nízká. Pro takové využití mého řešení by bylo nutné jej přepsat do jazyku C++.

Pokud je vstupním souborem do mého programu video, tak jsem objevil pár situací, kdy můj program nefunguje úplně správně. Jednou z takových situací je průjezd oblastí se stínem. V takovém případě můj program nedokáže správně detekovat silniční čáry, kvůli snížené intenzitě světla. Další takovou oblastí je silnice se zaprášenými postranními čárami. K tomu aby program detekoval takovou čáru je nutné snížit minimální a maximální práh detektoru hran, což ale vede ke zvýšenému počtu detekovaných hran, které už nutně nemusí souviset s jízdním pruhem.

Literatura

[1] https://towardsdatascience.com/lane-detection-with-deep-learning-part-2-3ba559b5c5af

[2] Gang Yi Jiang, Tae Young Choi, Suk Kyo Hong, Jae Wook Bae and Byung Suk Song, "Lane and obstacle detection based on fast inverse perspective mapping algorithm,"Systems, Man, and Cybernetics, 2000 IEEE International Conference on, Nashville, TN, 2000, pp. 2969-2974 vol.4.

[3] Wang, Jun & Mei, Tao & Kong, Bin & Wei, Hu. (2014). An approach of lane detection based on Inverse Perspective Mapping. 2014 17th IEEE International Conference on Intelligent Transportation Systems, ITSC 2014. 35-38. 10.1109/ITSC.2014.6957662.

[4] https://towardsdatascience.com/lane-detection-with-deep-learning-part-1-9e096f3320b7

[5] https://blogs.mathworks.com/deep-learning/2017/11/17/deep-learning-for-automated-driving-part-2-lane-detection/

[6] http://yann.lecun.com/exdb/publis/pdf/lecun-88.pdf

[7] http://cmp.felk.cvut.cz/ matas/papers/matas-bmvc98.pdf

[8] B. Yu and A. K. Jain, "Lane boundary detection using a multiresolution Hough transform,"Proceedings of International Conference on Image Processing, Santa Barbara, CA, 1997, pp. 748-751 vol.2. doi: 10.1109/ICIP.1997.638604

[9] J. M. Á. Alvarez and A. M. Lopez, "Road Detection Based on Illuminant Invariance,"in IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 1, pp. 184-193, March 2011.

[10] DOJAVA, M. Detekce jízdních pruhů a překážek [online]. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. 2011.

[11] http://journals.sagepub.com/doi/10.5772/54359

[12] X. Tan and B. Triggs, "Enhanced Local Texture Feature Sets for Face Recognition Under Difficult Lighting Conditions,"in IEEE Transactions on Image Processing, vol. 19, no. 6, pp. 1635-1650, June 2010.

[13] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection,"2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 779-788.

[14] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,"in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, June 1 2017.

[15] https://opencv.org/

[16] http://dlib.net/

[17] https://github.com/georgesung/road_lane_line_detection/blob/master/test_images/solidWhiteRight

[18] http://www.rszk.cz/aktual10/tz1035b.jpg

[19] https://i.ytimg.com/vi/AWbufP-7mr8/maxresdefault.jpg

[20] http://www.wallpapers4u.org/wp-content/uploads/road_turn_mountains_asphalt_tops_6353_1920x
450x253.jpg

[21] https://pixabay.com/en/road-drive-curve-roadtrip-signs-863126